 DOI 10.1109/MCSE.2016.16

Hardware Acceleration for Query Processing: Leveraging FPGAs, SIMD CPUs and 3D Memories

Oriol Arcas-Abella, Adrià Armejach, Timothy Hayes, Gorker Alp Malazgirt, Oscar Palomar, Behzad Salami, Nehir Sonmez

Introduction

Database management systems have become an indispensable tool for industry, government and academia and form a significant component of modern data centers. They can be used in a multitude of scenarios including online analytical processing, data mining, e-commerce and scientific analysis. The rate at which new data is being produced is growing every year; in 2000 it was estimated that there were 800,000 petabytes of data stored collectively in the world and in 2020 it is estimated that there will be 35 zettabytes[1]. Given this exponential growth, there is a pressure on software and hardware developers to create data centers that can cope with the increasing requirements. We take a look at the organization of a modern relational database management system (DBMS) and propose optimizations and redesign for the storage access, memory and CPU.

Currently our proposals are distinct techniques however we envision a unified system where these optimisations can complement one another and work holistically. Figure 1 shows a high-level overview of our conceived platform with the various components of our system. Reconfigurable architectures like Field-Programmable Gate Arrays (FPGA) are good candidates for hardware accelerators; our objective is to bring this specialized computation closer to the storage devices to reduce the high overheads associated with data movement. FPGAs not only provide massive fine-grain parallelism but are also resilient to irregular computation and storage access patterns. In the CPU, we investigate the capability of single-instruction multiple-data (SIMD) multimedia instruction extensions to accelerate sort, an important database operator with a significant processing overhead. We find that current SIMD ISAs lack the semantics to capture much of the available data-level parallelism (DLP) efficiently. Based on this observation we propose new SIMD instructions to capture irregular DLP and apply this to a novel sorting algorithm. In the memory hierarchy we focus on 3D stacked DRAM caches. This technology allows the integration of large amounts of memory with the processor die, increasing memory bandwidth and lowering access latency for the stacked DRAM. We investigate schemes to improve data placement in such caches in order to improve performance.

[1] 1 zettabyte is 1000^7 bytes or 1 trillion gigabytes

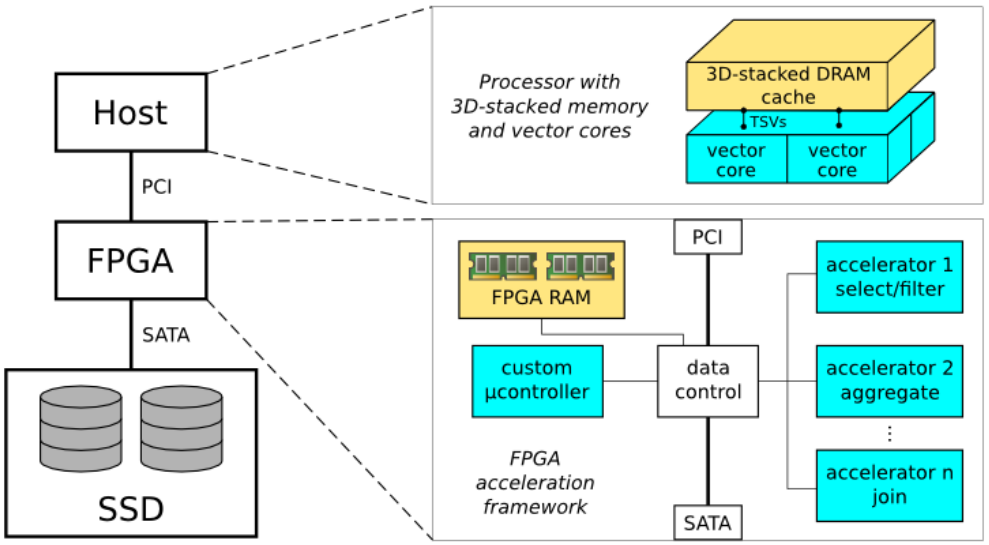


Figure 1: A high-level overview of the system. The host contains the CPU with vector extensions and 3D-stacked DRAM. Its duty is to run the DBMS software and selectively offload tasks to the acceleration framework. The acceleration framework has a high-bandwidth channel with the disk system and routes relevant data directly to the accelerator units or bypasses the request to the host.

FPGA-Based Acceleration Framework

We are developing an FPGA-based acceleration framework. The framework contains a dynamic dataflow-like network between the disk system and various accelerators implementing database primitives. Building upon this framework, we are investigating the potential of running entire complex DBMS queries precompiled on the board as well as the applicability of using high-level synthesis languages to allow novice programmers to create their own custom queries to execute directly on the framework.

Near-Data Processing

As the amount of data to manage grows, there is an increasing strain on DBMS software to meet its throughput and latency requirements. Data movement is a point of concern for modern data centers. Excessive movement can degrade both throughput and latency in addition to increasing the average energy cost per query. Near data processing aims to bring the computation closer to where the data resides so that more operations can be completed avoiding nonessential data movement. Although this is not the first study to process queries directly from disk using FPGA technology [1], our focus is on designing state of the art accelerators inside such an infrastructure. In this article, we will specifically focus on our hash join engine.

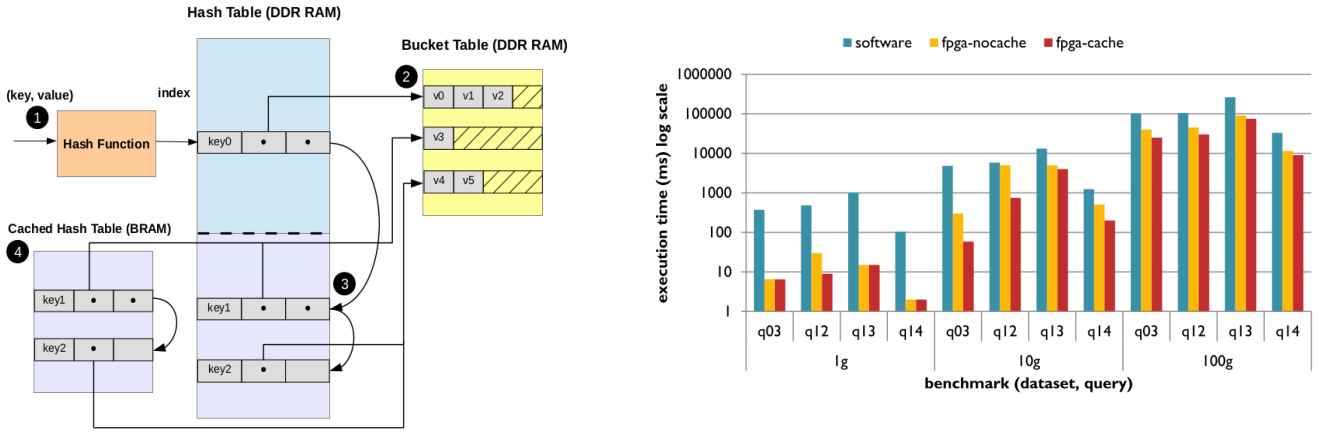
Figure 1 details our FPGA acceleration framework which is self-contained on an VC709 FPGA development board with a Virtex-7 FPGA and 8GB of RAM. The acceleration framework is in close proximity to a pair of 250GB SSDs. The host accesses the acceleration framework through a software API. In a typical SQL DBMS a query execution planner does the job of establishing an ordered set of steps that are used to access and process data. We have adapted this planner to appropriately use our infrastructure and to send and receive data and commands through a fast PCI link in order to communicate with the acceleration framework's custom microcontroller. Depending on the query plan that the

DBMS establishes, the host machine effectively programs the microcontroller to utilize the accelerator modules. This specialized microcontroller calculates the regions of the disk affected and instructs the appropriate accelerators to retrieve and process the data. The results can be stored back to the disk or sent to the host.

Query Processing Accelerators

We use Xilinx Vivado HLS to perform the selection/filtering operations. For this purpose, we have designed a compute engine that can process arbitrary-length data types in parallel. It takes rows as inputs and a condition, such as the BETWEEN operator. It applies a selection/filtering operation on the desired columns as they are read and filters out unwanted data from further processing, thus reducing the size of the input set [2].

In order to support join acceleration, we propose a novel architecture of hash join which is one of the most commonly used and time consuming DBMS operations [3]. Hash collisions, i.e. multiple distinct keys resolving to the same location in the hash table, are a critical issue that can be detrimental to good performance. Pointer chasing is a common method to resolve collisions however in DDR-based engines, the memory wall can undermine performance significantly. In order to overcome the memory wall, one feasible solution is to use the low-latency local Block RAMs (BRAMs) of FPGAs. But due to their small size there is no guarantee to store an entire hash table. Therefore we introduce a novel DDR-based hash join engine that transforms the BRAMs into a low latency cache. The entries of the main hash table are cached in the BRAM using a direct mapped methodology. The block diagram of the engine is shown in Figure 2(a).



(a) **(b)**

Figure 2: (a): An overview of our cached hash join engine; 1- compute hash index using a pipelined hash function. 2- grouping values of each individual key into the bucket table. 3- collisions are resolved with a pointer chasing method. 4- BRAM works as a cache of the main hash table. (b): The execution time of cache-employed version compared to the baseline and software.

We developed the fully pipelined version of the proposed hash join architecture using BlueSpec SystemVerilog HLS and performed experiments on a selection of TPC-H queries. Depending on the BRAM hit rate, experimental results show up to 2.8x improvement in the number of cycles over an FPGA baseline without BRAM and up to 82.3x over the hash join engine of PostgreSQL running on RAMDisk. Detailed experimental results are shown in

Figure 2(b) for queries q03, q12, q13, q14 of TPC-H running three dataset sizes: 1g, 10g and 100g [3].

Even though we have developed accelerators for fundamental database primitives, further performance improvements are possible if the framework provides acceleration for user-specific queries that may require customized processing not easily expressible through default operators. As the optimal accelerator would be very much workload dependent, the implementation would depend on the DBMS user, who is typically not an expert in FPGA development. We have studied four high-level synthesis tools that may help non-expert users to accelerate common database operations [4]. Choosing the right design languages and tools can greatly affect the performance of the final hardware. Our results show that high-level hardware description languages can produce accelerators with similar performance to manually designed hardware. In our study, we implement four accelerators for database operations. The programming experience varies greatly from one design paradigm to the other. Advanced High-Level Synthesis (HLS) tools, which can generate hardware from pure C code, are still in the initial stages although they look highly promising as a solution for non-expert users to generate high-quality designs with little to no effort.

Novel SIMD Instructions for Sorting

FPGA-based acceleration can be used to offload many tasks from the main processing unit; nevertheless, the CPU still plays a crucial role. Modern CPUs can yield very high performance if the algorithm is carefully tuned and leverages the architecture features thoroughly. As a case study, we focus on sorting and vector SIMD extensions, one of the most efficient and powerful features available in commodity processors and essential to achieve high performance on the CPU.

Sorting is a cornerstone of relational databases. It is a frequently used primitive operation and often used in the construction of composite functions. Given this operation's importance, it is vital to have a high performance algorithm. By leveraging the SIMD extensions in modern microprocessors, sorting could potentially take advantage of explicit DLP and provide more bandwidth between the functional units and the memory hierarchy. SIMD extensions have become ubiquitous in modern microprocessors, however as it stands today, current extensions lack essential features that inhibit scalability and speed when vectorizing sorting algorithms. We anticipate that SIMD extensions will grow both in width and functionality in future generations and our contribution may help guide this evolution in a clear and positive way. In Figure 1, each CPU core is augmented with sophisticated SIMD support reminiscent of a vector architecture.

For relational databases, radix sort is arguably the best choice of sorting algorithm. Primary and foreign keys have a finite radix and a cardinality well below their respective datatype's limit. Previous attempts at vectorizing radix sort with a SIMD ISA have been only semi-successful. The reasons are numerous but boil down to two fundamental problems: (1) radix sort has strict stability requirements meaning it is a serial algorithm in nature. (2) Actions within standard SIMD operations should be independent of one another. This complicates the process of updating monolithic bookkeeping structures such as histograms as there are often conflicting gather-modify-scatter operations. To circumvent this irregular

DLP, significant transformations must be made to radix sort in order to vectorize it. The input needs to be read incrementally using an inefficient strided memory access pattern so that individual elements of a vector register operate on contiguous partitions, and bookkeeping structures need to be replicated for every element in a vector register to avoid update conflicts.

Based on these observations we propose VSR sort [5], a novel way of vectorizing radix sort without these inefficient transformations. We introduce two new instructions into our SIMD ISA. Vector Prior Instances (VPI) allows us to load the input and store the output using an efficient unit-stride (contiguous) memory access pattern without breaking the strict stability requirements of radix sort. In essence, it corrects problematic input values that have a serial dependency and ensures they end up in adjacent locations of the output. Vector Last Unique (VLU) can be used to update non-replicated bookkeeping structures while avoiding gather-modify-scatter conflicts. Using non-replicated bookkeeping structures allows us to grow them considerably and consequently process a larger subset of bits per pass thereby reducing the runtime of the algorithm. The full semantics and implementation of these instructions is beyond the scope of this article and the reader referred to [5] for further details.

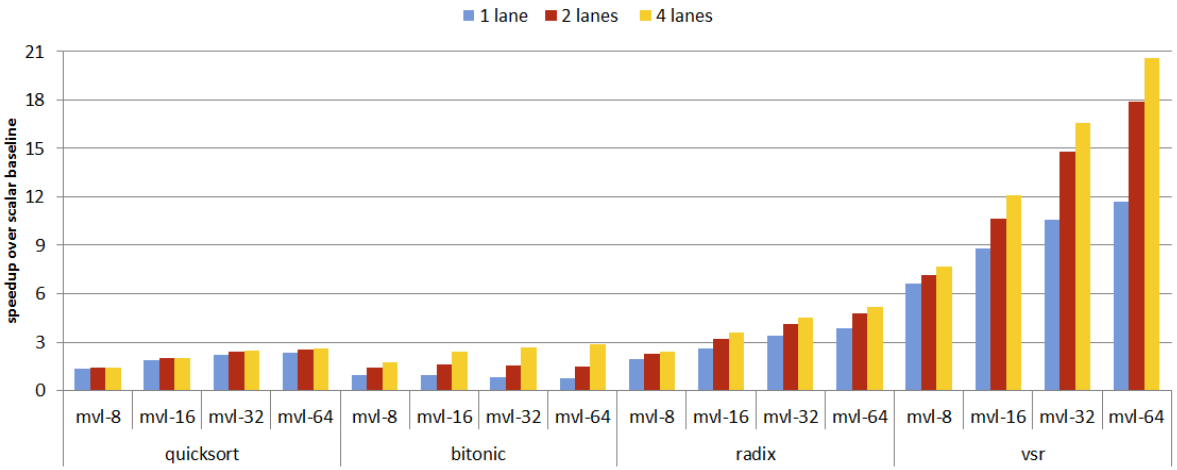


Figure 3: Speedup over scalar algorithm for vectorized versions of quicksort, bitonic mergesort, radix sort and our novel VSR Sort algorithms while varying the maximum vector length (mvl) and vector lanes.

Using a custom simulation framework, we make significant sensitivity experiments of the performance and scalability of VSR sort and three diverse SIMD sorting algorithms suitable for relational databases: quicksort, bitonic mergesort and radix sort. We take into account our predictions of width and functionality in future microprocessor generations. Figure 3 shows a comparison of each algorithm The metric used is speedup over a scalar baseline. Here we change two critical parameters: the *maximum vector length (mvl)* which is the number of elements in a SIMD register; and *lanes*, which refers the number of redundant functional units available to a single SIMD instruction. A configuration with a single lane resembles a purely pipelined functional unit implementation like that of the CRAY-1. We find that all the prior work suffers from bottlenecks and scalability problems. VSR sort exhibits good scalability and achieves maximum speedups up to 20.6x over a scalar baseline and on

average performs 3.4x better over a standard SIMD radix sort when run on the same hardware configuration. We feel that VSR sort is an excellent candidate to implement sort primitives in a relational database.

Leveraging Emerging Memory Technologies

Due to the ever increasing computational throughput of chip multiprocessors, off-chip memory has become a performance-limiting factor due to limited pin count scalability. Current off-chip memory bandwidth capabilities are not sufficient to meet the demands of modern servers running data demanding workloads such as those employing relational databases [6]. However, die-stacking technology enables tightly coupled integration of DRAM with the processor die - as can be seen in Figure 1 - providing hundreds of megabytes to several gigabytes of storage that can be accessed at an unprecedented bandwidth with latencies that are significantly lower than those needed to access off-chip memory. Unfortunately, the amount of storage that this technology offers is still not sufficient to cope with memory capacity requirements, which are often one or two orders of magnitude higher [7]. For this reason, researchers have been exploring the use of stacked DRAM as a large last-level cache.

We have analyzed current DRAM cache proposals and identified different performance pathologies that arise due to interleaved accesses from different cores at the DRAM banks. Figure 4 illustrates two of these pathologies that have a significant impact on performance: (a) row-buffer interference and (b) utility interference. The figure shows different queued requests for each bank, and for each request, the requesting core and the targeted row (i.e. page) within the DRAM bank where the data is known to be present.

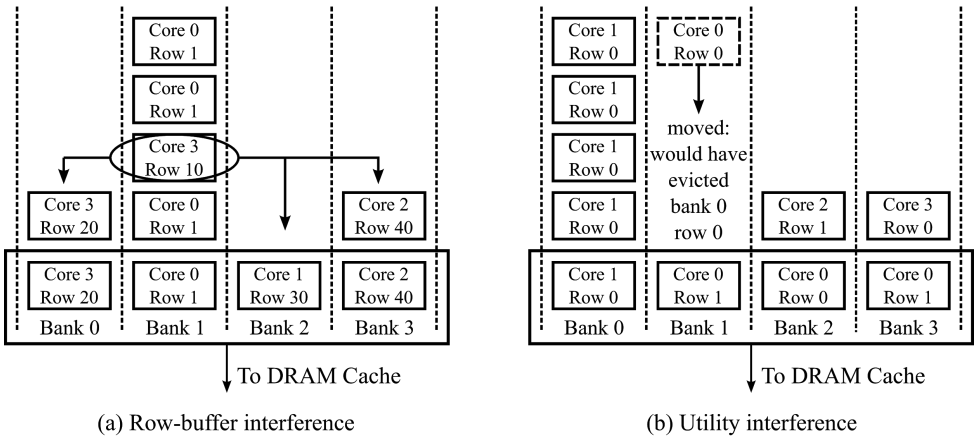


Figure 4: Performance pathologies present in DRAM caches.

Figure 4a illustrates a case of row-buffer interference where $core_0$ is performing a number of accesses over the same DRAM row (page) on $bank_1$, for which the access latency is low due to row-buffer hits. However, there is a request to a different row from $core_3$ that needs to be served, destroying this row-buffer locality. We would like to avoid any interference from other cores that may prematurely close the row-buffer just to service a single request, having to open again the previous row to finish reading it.

Figure 4b illustrates a case of utility interference. In the example, $bank_0$ contains a page accessed by $core_1$ with high spatial locality, from which many blocks are useful - i.e. demanded. The rest of the cores are accessing pages with low spatial locality, which suggests a lower number of useful blocks. If the incoming request in $bank_1$, which was a cache miss, had been allocated in $bank_0$, then the page with high locality and block usage would have been evicted even if there were better replacement candidates in other banks. We would like to protect cores that would experience significant performance improvements from interfering cores that might hinder these improvements by thrashing the cache - evicting highly reused blocks, undermining cache utility.

In the context of database workloads, memory bandwidth and access latency are critical to achieve good performance. Moreover, in such workloads, there is abundant spatial locality present in a significant number of common primitives, such as sequentially reading from a table. While a DRAM cache has the potential to be a great performance booster for these kind of workloads, the mentioned performance pathologies degrade the potential performance improvements. As Figure 4 suggests these pathologies can be mitigated with a better data placement within the cache. Our proposal aims at providing this functionality, improving DRAM cache performance [8].

In particular, our proposal identifies cores experiencing high spatial locality and dynamically modifies the DRAM cache replacement policy to allocate their pages on a subset of the banks. This data placement policy reduces row-buffer interference by avoiding to close the row-buffer to service a single access in the common case, since high spatial locality implies high row-buffer locality. Similarly, allocations for cores with poor spatial locality are placed on another subset of the banks, where row-buffer locality is less penalized because it is already low. In addition, utility interference is also reduced because rows with a high block demand will not be replaced for rows with low demand. Banks where low spatial locality cores allocate pages are likely to generate more evictions, but these evicted pages are cheaper to fetch back and have a lower impact on the overall cache hit ratio.

We evaluate our proposal using an architectural simulator that models a chip multiprocessor (CMP) with 8 out-of-order cores and a 3-level cache hierarchy, which closely resembles a server grade Sandy Bridge Intel processor. To model the stacked and the off-chip DRAM memory we have integrated a detailed main memory simulator. For this study we have configured our DRAM cache to have a size of 2GB and four memory channels. As a representative data analytics workload we use a set of queries (Q9, Q13, Q15 and Q16) from the TPC-H benchmark running on a modern column-store database engine with a dataset that exceeds 100GB. Our proposal yields a 24.5% performance improvement over a system without stacked DRAM, and outperforms a state-of-the-art DRAM cache proposal [9] by 10%.

Conclusion

In this work we have proposed hardware improvements for DBMSs using emerging technologies: FPGAs, future SIMD support and die-stacking DRAM. We anticipate that applying these redesigns to all levels of the system will result in data centers which can provide a many-fold increase in throughput. Moreover, an important aspect often dismissed

by hardware researchers is programmability and accessibility; we present evaluation and discussion related to HLS applicability to novice programmers to create custom DBMS functionality. Our goal is to bridge the gap between these specialized technologies and the non-expert end user.

Although our work currently comprises several disparate ideas, our ultimate goal is to bring these together in a unified system and allow them to cooperate harmoniously. Vector SIMD extensions are a good fit for certain key algorithms, however its performance is typically limited by the available memory bandwidth. A die-stacked DRAM cache provides abundant memory bandwidth which can unlock additional performance for vector extensions while providing the additional benefits described in this paper. While these two technologies work well in tandem, data movement between storage systems and processing units is a major bottleneck that will be exacerbated as the amount of data to be processed grows. To minimize data movement we propose to use FPGA technology for near-data processing and to tackle certain operations that map well to these devices, reducing data movement to computational resources.

Acknowledgements

The research leading to these results has received funding from the European Union's Seventh Framework Programme (FP7/2007- 2013) under the AXLE project (GA no 318633).

References

[1] Scofield, Todd C. et al. "XtremeData dbX: An FPGA-Based Data Warehouse Appliance." Computing in Science and Engineering (CiSE), 2010.

[2] Malazgirt, Gorker Alp, et al. "Accelerating Complete Decision Support Queries Through High-Level Synthesis Technology." Field-Programmable Gate Arrays (FPGA), 2015.

[3] Salami, Behzad, et al. "HATCH: Hash Table Caching in Hardware for Efficient Relational Join on FPGA." Field-Programmable Custom Computing Machines (FCCM), 2015.

[4] Arcas-Abella, Oriol, et al. "An empirical evaluation of High-Level Synthesis languages and tools for database acceleration." Field Programmable Logic and Applications (FPL), 2014.

[5] Hayes, Timothy, et al. "VSR sort: A novel vectorised sorting algorithm & architecture extensions for future microprocessors." High Performance Computer Architecture (HPCA), 2015.

[6] Rogers, Brian M, et al. "Scaling the bandwidth wall: challenges in and avenues for CMP scaling." International Symposium on Computer architecture (ISCA), 2009.

[7] Ferdman, Michael, et al. "Clearing the clouds: a study of emerging scale-out workloads on modern hardware." Architectural Support for Programming Languages and Operating Systems (ASPLOS), 2012.

[8] Armejach, Adrià, et al. “Tidy Cache: Improving Data Placement in Die-stacked DRAM Caches.” International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD), 2015.

[9] Jevdjic, Djordje, et al. "Die-Stacked DRAM Caches for Servers." International Symposium on Computer Architecture (ISCA), 2013

Bios

Oriol Arcas-Abella is a PhD candidate in computer architecture at the Universitat Politècnica de Catalunya, where he obtained his BS and MS degrees in computer science in 2010. His research in the Barcelona Supercomputing Center is focused on soft microarchitecture prototyping on reconfigurable devices, including hardware debugging and verification techniques and high-level design. Contact him at oriol.arcas@bsc.es.

Adrià Armejach is a post-doctoral researcher at the Barcelona Supercomputing Center. His interests include computer architecture, parallel computing, memory systems, and performance evaluation. He received a PhD from the Universitat Politècnica de Catalunya (UPC) in 2014 and is currently involved in providing solutions using emerging memory technologies in the context of an FP7 project on Advanced Analytics for Extremely Large European Databases (AXLE). Contact him at adria.armejach@bsc.es.

Timothy Hayes is a third year PhD candidate in the field of Computer Architecture at the Barcelona Supercomputing Center. His doctoral thesis centres around the creation of novel vector architectures for data management. His research interests include SIMD models of computation, workload characterisation and high-performance microarchitecture development. Contact him at timothy.hayes@bsc.es.

Gorker Alp Malazgirt is a PhD candidate in Computer Engineering at Bogazici University, Istanbul. He was a recipient of Severo Ochoa mobility grant from Barcelona Supercomputing Center during the summer of 2014. He received his BSc. in Microelectronics Engineering from Sabanci University, Turkey, and MSc in System-On-Chip design from Lund University, Sweden. His research interests are computer architectures, reconfigurable computing and metaheuristics. Contact him at alp.malazgirt@boun.edu.tr.

Oscar Palomar earned his PhD in Computer Architecture in 2011 from the Universitat Politècnica de Catalunya. Since 2010 he has been working at the Barcelona Supercomputing Center in the Computer Architectures for Parallel Paradigms group. His research interests involve low-power vector architectures and energy minimization. Contact him at oscar.palomar@bsc.es.

Behzad Salami holds BSc degree in Computer Engineering from Iran University of Science and technology (IUST) and MSc from Amirkabir University of Technology (AUT). He is pursuing a PhD degree, at University Politècnica de Catalunya (UPC), while doing research at Barcelona Supercomputing Center (BSC). His research interests are reconfigurable computing, big data processing and high performance computing. Contact him at behzad.salami@bsc.es.

Nehir Sonmez holds a PhD in Computer Engineering from the Universitat Politècnica de Catalunya, an MSc from Bogazici University and a BSc from Syracuse University. He is currently a post-doctoral researcher in the Computer Architecture for Parallel Paradigms research group at the Barcelona Supercomputing Center. His research interests include reconfigurable computing, multicore computer architecture and database acceleration for big data and transactional memory. Contact him at nehir.sonmez@bsc.es.